# Interaction Styles in Tools for Developing Virtual Reality Applications

Kjeldskov, Jesper; Stage, Jan

*Published in:*
Human-Computer Interaction Research and Practice

*Publication date:*
2001

*Document Version*
Early version, also known as pre-print

# Interaction Styles in Tools for Developing Virtual Reality Applications

*Jesper Kjeldskov*
Aalborg University
Department of Computer Science
Fredrik Bajers Vej 7
DK-9220 Aalborg East, Denmark
jesper@cs.auc.dk

*Jan Stage*
Aalborg University
Department of Computer Science
Fredrik Bajers Vej 7
DK-9220 Aalborg East, Denmark
jans@cs.auc.dk

**SUMMARY**
Virtual reality display systems reflect an emerging technology that is used to visualize virtual three-dimensional (3D) worlds. This article describes and evaluates tools for developing software applications that are targeted at virtual reality display systems. The evaluation focuses on the relevance of command language or direct manipulation with graphical representation as the fundamental interaction style in such a tool. The foundation of the evaluation is an empirical study of two development processes where two tools representing each of these interaction styles was used to develop the same virtual reality application. The development tool that employed a command language was very flexible and facilitated an even distribution of effort and progress over time, but debugging and identification of errors was very difficult. The tool that employed direct manipulation enabled faster implementation of a first prototype but did not facilitate a shorter implementation process as a whole due to e.g. lack of support for writing code for repetitive operations.

**KEYWORDS:** virtual reality, new user interfaces, interaction styles, direct manipulation.

## INTRODUCTION

Methods and guidelines for user interface design embody certain computer technologies. This also applies to interaction styles. Interaction based on a command language was a relevant solution with the character-based display. Direct manipulation emerged from the potentials of the graphical workstation and personal computer. This inherent relation between interface design and computer technology implies that our established guidelines and experiences are challenged when new technologies emerge.

Virtual reality display systems are used to create and visualize virtual three-dimensional (3D) worlds. This technology is emerging, and practically relevant applications are being developed for a broad range of domains. As the use of the technology is increasing, there is an increasing demand for tools for developing applications.

The purpose of this article is to compare and discuss the relevance of classical interaction styles for tools that are used to develop virtual reality applications. The focus is on the process of developing actual virtual reality applications. More specifically, we compare the potentials of a development tool based on a command language with one that is based on direct manipulation. The discussion is structured in the following way. In the first section, we review selected literature on command language and direct manipulation as classical interaction styles. We then describe the virtual reality display technology in order to specify the target platform we are aiming at. The comparison of the two classical interaction styles is based on an empirical experiment. In the following sections, we describe the design of that experiment and emphasize the relevant results. Finally we conclude the discussion and point out avenues for further work.

## INTERACTION STYLES IN DEVELOPMENT TOOLS

The interaction style is a key determinant of the design of the user interface. Many discussions on the advantage or disadvantage of a certain design relate to this characteristic. The options available for design of this characteristic have been denoted as: command language, menu selection, form filling, and direct manipulation [10]. Below, we will refer to these as the classical interaction styles.

The development of a virtual reality application includes an essential task where we construct the 3D world that is visualized by the application. The fundamental interaction style of existing tools that support this task employs either direct manipulation or a command language in the form of a textual programming language, cf. section 3. Menu selection and form filling are also employed but only for secondary interactions that deal with limited issues, e.g. the specification of properties of a certain object that is manipulated directly on an overall level. Based on these priorities, the discussion below deals only with command language and direct manipulation.

The literature on human-computer interaction includes numerous contributions to the question whether direct manipulation is superior to command languages. Much of this is description of advantages whereas the amount of empirical evidence is very limited [2]. An early contribution focusing directly on construction compared file manipulation commands in MS-DOS with Macintosh direct manipulation. This study concluded that the Macintosh users could perform the manipulations faster, with fewer errors, and they were more satisfied with the interface [6]. A similar study where command line and direct manipulation was compared concluded that the users of direct manipulation made only half as many errors and were more satisfied. In this study, the time to perform the tasks turned out to be comparable [7].

The limited number of reports from empirical studies of command language and direct manipulation seem to indicate an advantage in terms of error rate and user satisfaction. When it comes to time to complete a task, the conclusions are more varied.

Our intention in this paper is to examine these expectations in relation to tools for developing virtual reality applications.

## VIRTUAL REALITY APPLICATIONS

A virtual reality application that visualizes a 3D world consists of a number of mathematically defined 3D models that are covered with colors or textures, e.g. pictures or video images. The 3D models are spatially distributed in a three-dimensional coordinate system that the user can experience as a 3D world by viewing the 3D models from a given point in the coordinate system. The correct perspective is rendered real-time by a graphics computer and projected by means of a display system as illustrated in figure 1. Motion in the virtual world is accomplished by changing viewpoint by either means of position tracking or a specialized interaction device. Interaction with objects in the virtual world is typically supported by techniques for selecting and modifying 3D objects by simply "grabbing" them just like one would do in the real world. The 3D experience requires shutter glasses worn by the user allowing separate images to be projected to the user's left and right eye and thereby creating an illusion of 3D. Tracking the position of user's head ensures that the correct visual perspective is calculated.

A virtual reality application may use a multitude of display systems to visualize the virtual 3D world. Examples of display systems are traditional desktop monitors, head-mounted displays, holobenches, large wall-mounted displays or caves with various numbers of sides. These display types represent the array of technologies for creating immersive experiences that range from "looking at" a virtual 3D world to "being in" that virtual world [10].


***Figure 1:*** A virtual 3D world

The six-sided cave (Cave Automated Virtual Environment) is currently the display system that offers the greatest level of immersion into a virtual 3D world. The user is placed in a small cubic room, measuring approx. 3 meters on all sides, in which the computer-generated images are back-projected on all four walls, the floor and the ceiling, cf. figure 2.

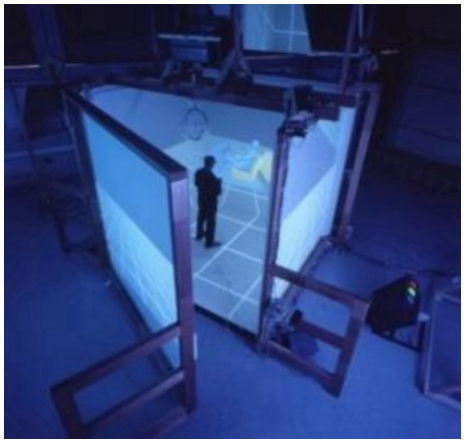

***Figure 2:*** Outside the six-sided cave

The benefits of the six-sided cave for exploration of virtual 3D worlds lay in the vividness of the virtual environment projected and the very high degree of immersion. This is caused by the freedom of movement that is possible inside the cave and the large horizontal and vertical field of view covered with images. Exploration of the virtual world is much more natural in a six-sided cave compared to any other display system because the user can move around physically and look in any direction without breaking the illusion of being in a computer-generated world. The primary downside is that physical objects and the user's body itself may occlude the images, thus locally breaking the visual illusion.

***Figure 3:*** Inside the six-sided cave

Virtual reality applications displayed in a cave are very different from many traditional computer applications. First, the user interface is completely surrounding the user and is presented in 3D as opposed to conventional 2D interfaces covering only a fraction of the user's physical surroundings. Second, the types of applications running in a cave are typically offering a complete virtual 3D world for exploration as opposed to traditional tools for office or home use. Third, applications running in a cave are by default both highly graphical and interactive.

## TWO CATEGORIES OF DEVELOPMENT TOOLS

Although virtual reality applications are fundamentally different from typical applications, they are not developed *in* the cave itself. Virtual 3D worlds are usually developed on ordinary – yet powerful – desktop computers with traditional 2D displays. The existing tools for development of virtual reality application fall in two different categories.

The first category can be characterized as a classical programming approach, since the creation and manipulation of the virtual world and the objects in it is specified in a command language. Within this approach, special libraries for creating cave applications are available for C and C++. One of the most widely used binary libraries for developing virtual 3D worlds is CaveLib. This library enables development of highly immersive 3D interfaces for projection in a cave, or any other virtual reality display system, as well as implementation of interaction techniques for 3D interaction devices. For preview purposes, CaveLib offers a simple tool for representing the cave display and simulating simple 3D interaction, cf. figure 4.

Using CaveLib to develop an application is not very different from developing any other graphical application in a typical programming language.

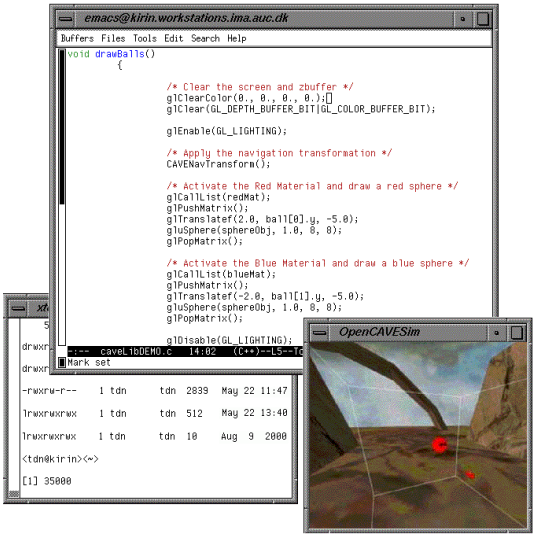


***Figure 4:*** Development with CaveLib

The second category of tools for developing virtual reality applications can be characterized as a graphical representation and direct manipulation approach. One of the few professional tools in this category is dvMockup.

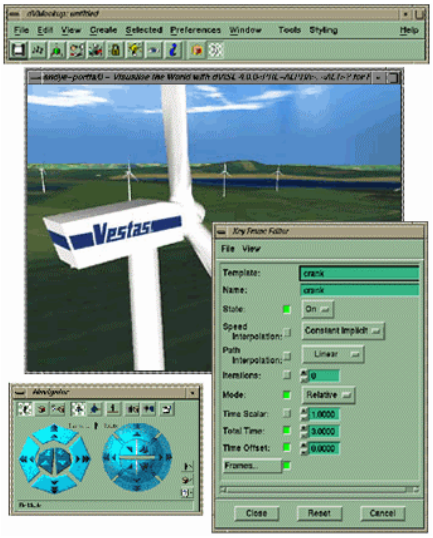


***Figure 5:*** Development with dvMockup

This tool enables the developer to create an application by directly manipulating the objects of the virtual 3D world within the preview window along with the use of menu selections and fill-in forms, cf. figure 5.

Using dvMockup, implementing an application for the cave is done without doing any actual programming.

## EXPERIMENTAL DESIGN

An empirical experiment was conducted to inquire into the research question that was raised in section 1. This section describes the design of that experiment.

*Tools:* We briefly surveyed potentially relevant tools for implementing virtual reality applications and related this to the fundamental aim of comparing direct manipulation tools with programming tools. Based on the survey and the facilities available, we selected two established tools: dvMockup, a direct manipulation tool that enables people without programming experience to create a virtual reality application, and CaveLib, an advanced programming tool that facilitates development of virtual reality applications characterized by high performance and flexibility. In addition, we selected a new and promising programming tool, VR Juggler, which extends CaveLib with a more flexible and modular structure and open source architecture. The first two tools were already installed, configured, and used extensively by other developers and researchers who could be consulted when technical problems arose. The third tool, VR Juggler, was acquired right before the beginning of the experiment and there were no experiences with it.

*Participants:* A development team of three persons and the two authors of this article planned and designed the experiment, whereas the development team conducted the implementation phase. The three developers had recently completed a master degree in computer science/computer engineering. Thereby, they had considerable experience and knowledge about programming in general. They had previously taken a one-semester course on computer vision and virtual reality and worked with projects within that subject. They received a one-day introduction to the tools used in the experiment but had no experience with them.

*Overall task:* The comparison of the three tools was based on solution of the same overall task. The overall task was to develop a virtual reality application that visualized a maze in which a user could move an avatar around by means of an interaction device. This task was specified in detail by dividing it into 14 milestones. Thus, the overall task was solved when all milestones were met. The milestones involved tool and cave set-up (milestone 1 and 2), implementation of a simple application (milestone 3 and 4), implementation of the application visualizing the maze (milestone 5 to 8), interaction techniques to facilitate motion of the avatar (milestone 9 to 12), and adjustment (milestone 13 and 14).

*Hypothesis:* Based on the literature on interaction styles reviewed in section 2 above, we developed the following hypothesis: The direct manipulation tool is superior to the programming tools in terms of the efforts required to implement the a virtual reality application that is specified by the overall task.

*Experimental procedure:* When the planning phase of the experiment was complete, the implementation phase started. This phase was planned to last three weeks but was extended with a couple of days because of technical problems. Each member of the development team was assigned one of the three tools and should use it to produce the best possible solution to the overall task. During the implementation phase, they were not supposed to communicate with each other about their work, problems, and solutions.

*Data collection:* The primary means for data collection were private diaries written by each developer, cf. [4], [8]. After a day of work on the implementation, each developer used about an hour to describe the work done and its relation to the 14 milestones, the problems faced, and the time spent on tasks related to each of the milestones. A checklist that emphasized the points that should be touched upon supported the daily writing of the diary. One week into the implementation phase, the diary entries produced so far were reviewed enforces the use of the checklist and increase consistency. The three diaries amount to a total of 45 pages [3].

*Data analysis:* The primary dependent variables were work practice and development effort. Yet, in this article we focus only on development effort. Based on the diaries, we have calculated and compared the efforts spent on completing the different milestones of the overall task. The results of this are presented in the following section.

*Limitations:* The design of this experiment imposes certain limitations on our results. Firstly, the members of the development team were not highly experienced in implementing virtual reality applications. Secondly, The overall task defined a specific application that should be implemented. These limitations imply that the results primarily facilitate relative as opposed to absolute conclusions about efforts. Thirdly, the diaries of the three developers were different. In order to handle this they were reviewed after one week of work. The fourth limitation was that two of the tools were established on the development platform whereas the third was not even configured and there was no experience with its use. The developer who worked with this tool ended up spending a considerable amount of effort on issues related to installation and execution. Therefore, we ignore this part of the experiment in our comparison below.

## FINDINGS

In this section, we present and discuss the findings from the experiment with CaveLib and dvMockup. The developer who used CaveLib was able to meet all milestones, but the navigation technique specified was changed due to usability issues. The developer who

used dvMockup was not as successful, since collision detection could not be implemented satisfactory. However, the final solution was acceptable. The development time spent using CaveLib amounts to 42,3 hours, whereas the time spent using dvMockup amounts to 37,8 hours. The total time spent on development with the two tools thus differ only 12%. The distribution of time spent on each milestone does, however, reveal significant differences between the programming and direct manipulation approaches. This distribution is shown in figure 6. Below, we will highlight interesting points from this distribution.

Setting up the development tools and the cave (milestone 1 and 2) amounted to a total of 12 hours spent on CaveLib whereas only 3,8 hours was spent on this with dvMockup. Thus the developer who used dvMockup only needed about 30% of the time spent using CaveLib. Setting up CaveLib demanded a series of separate tools to be configured for individual tasks, e.g. scripting, compiling, and previewing, as well as creation of a number of configuration files on both the workstation used for development and the graphics computer that was executing the display system for the cave. With dvMockup only one tool had to be set up, and when an application was running on the workstation, only a few scripts were needed before it was also operational in the cave.

Implementation of a preliminary application with the purpose of testing the development and target platform and the connection between them (milestone 3 and 4) took 6,5 hours using CaveLib but only 2 hours with dvMockup. Again, for dvMockup this is only about 30% the time spent using CaveLib. Thus up to milestone 4 it is clear that the direct manipulation approach supports a faster kick-off on the development process

Implementation of the primary application, which was the maze specified in the overall task (milestone 5 to 8), was done in 10,3 hours using CaveLib. With dvMockup the same milestones required 27,5 hours. So here we see the same pattern where one tool requires only about 30% of the time spent with the other tool. Yet this time the roles are reversed, as CaveLib is the favored tool. Thus the programming approach seems to facilitate a more effective process in this part of the implementation. The major reason for the considerable amount of time spent with dvMockup is that the tool provides no direct support in a situation where making and running a simple program might avoid numerous repetitions of simple operations. For example, the developer using dvMockup faced the task of *manually* inserting 800 identical cubic 3D objects into the virtual 3D world, whereas the developer using CaveLib could perform the same task simply by writing a small piece of code. This limitation becomes even more serious when we take the question of scale into consideration. If we compare a small application to a large one, the difference in amount of work will occur precisely on milestone 5 to 8 whereas the remaining milestones will largely be unaffected. Therefore, the difference between the two tools on these milestones will even be more significant if we expand the scale of the application being developed.

Implementation of interaction techniques (milestone 9 to 12) took 7,5 hours with CaveLib and only 2,5 hours using dvMockup. This is a 30% reduction in favor of dvMockup.

The time spent implementing interaction techniques with dvMockup is, however, influenced by the availability of supporting software. In a related project, a considerable amount of time had been spent developing "off-the-shelf support" for implementing interaction in dvMockup in order to facilitate a general reduction of development time [5]. Had this support not been available, the time spent on these milestones would definitely have increased, but we will not attempt to estimate by how much. In CaveLib, all interaction techniques were implemented from scratch.

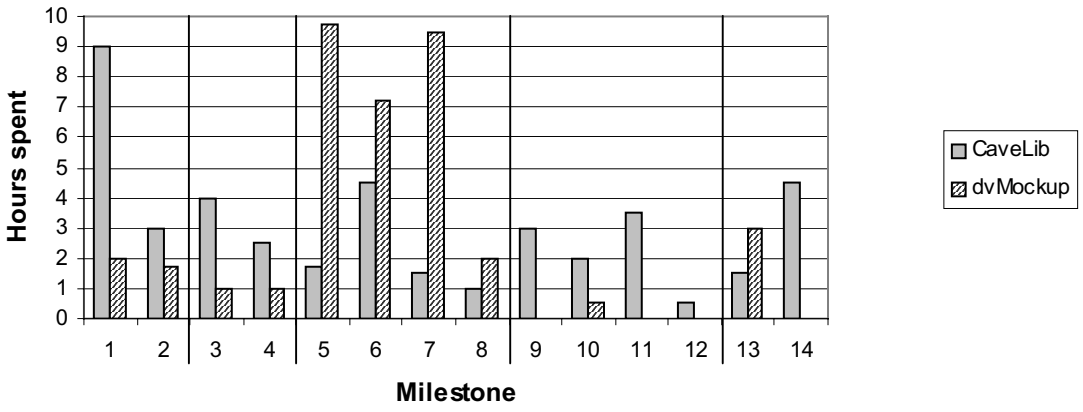


***Figure 6:*** Development time spent using CaveLib and dvMockup.

However, this had the advantage that the interaction technique specified in the overall task was actually implemented. With dvMockup it was necessary to select one of the available techniques, which did not fulfill the specification completely. If the implementation in dvMockup should have fulfilled the requirements completely, additional programming on device driver level would have been necessary.

Final adjustment of the applications (milestone 13 and 14) took 6 hours for CaveLib while only 3 hours was spent with dvMockup. The larger amount of adjustments of the CaveLib application primarily consisted of correcting errors with the scaling of 3D objects. This was necessary to make the objects fit properly for projections in the cave. This kind of errors was absent in the application developed with dvMockup.

## CONCLUSION

We have conducted a qualitative empirical study showing that implementing a virtual reality application using a command language tool and a direct manipulation tool required efforts in terms of time that are comparable. The command language tool, however, resulted in faster implementation during the most essential phases of the implementation process and thus outperforms the direct manipulation tool on a larger scale. The direct manipulation tool on the other hand resulted in fewer errors.

By focusing on application development for a six-sided cave using tools running on desktop computers we have, of course, taken things to the edge. There is a continuum of virtual reality displays for which the distance between development tool and target application may not be as significant as for the six-sided cave.

A central question rises from this conclusion. Can direct manipulation be further exploited in tools for developing virtual reality applications? A relevant solution might be to make direct manipulation more direct as discussed in [1] and [9].

## ACKNOWLEDGEMENTS

The authors thank the development team: Mike H. Hougaard, Nikolaj Kolbe and Flemming N. Larsen. We also thank VR-MediaLab for access to virtual reality installations and development tools.

## BIBLIOGRAPHY

1. Beaudouin-Lafon, M. (2000). Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. *CHI Letters,* 3(1): 446-453.
2. Benbasat, I. and Todd, P. (1993). An Experimental Investigation of Interface Design Alternatives: Icon vs. Text and Direct Manipulation vs. Menus. *International Journal of Man-Machine Studies*, Vol. 38, 1993, pp. 369-402.
3. Hougaard, M. H., N. Kolbe and F. N. Larsen (2001). *Comparison of Tools for Developing virtual reality Application.* (In Danish). Aalborg University: Intermedia.
4. Jepsen, L. O., L. Mathiassen and P. A. Nielsen (1989). Back to Thinking Mode: Diaries for the Management of Information System Development Projects. *Behaviour and Information Technology* 8(3): 207-217.
5. Kjeldskov, J. (2001). Interaction: Full and Partial Immersive virtual reality Displays. Accepted for publication at the 24th IRIS conference.
6. Margono, S. and Shneiderman, B. (1987). A study of File Manipulation by Novices Using Commands vs. Direct Manipulation. In *Proceedings of the 26th Annual Technical Symposium*, ACM, Washington, DC, 1987, pp. 57-62.
7. Morgan, K., Morris, R. L. and Gibbs S. (1991). When Does a Mouse Become a Rat? or … Comparing the Performance and Preferences in Direct Manipulation and Command Line Environment. *Computer Journal*, Vol. 34, pp. 265-271.
8. Naur, P. (1983). Program Development Studies Based on Diaries. In: T. R. Green et al. (Eds.), *Psychology of Computer Use*, pp. 159-170. London: Academic Press.
9. Schkolne, S., Pruett, M., and Schröder, P. (2001). Surface Drawing: Creating Organic 3D Shapes with the Hand and Tangible Tools. *CHI Letters,* 2(1): 261-268.
10. Shneiderman, B. (1998). *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison Wesley Longman, Reading, Massachussetts, 1998.